Faster & Greater Messaging System
HornetQ zzz

Giovanni Marigi
gmarigi at redhat.com
Middleware Consultant
JBoss, a Division of Red Hat

HornetQ

Agenda

- Intro
- Core
- EAP and standalone
- Transport
- Persistence & large messages
- Flow control
- Clustering & High Availability
- Other features

Some stats

HornetQ sets a record breaking score in the SPECjms2007 industry standard benchmark for JMS messaging system performance.

HornetQ 2.0.GA obtained scores up to 307% higher than previously published
SPECjms2007 benchmark results, on the same server hardware and operating system set-up.

The peer-reviewed results are available on the spec.org web-site:
www.spec.org/jms2007/results/jms2007.html

8.2 messages per second with SpecJMS
http://planet.jboss.org/post/8_2_million_messages_second_with_specjms

The results were obtained by Kai Sachs and Stefan Appel from an independent research group at the TU Darmstadt, Germany.

Their release announcement can be found here:

www.dvs.tu-darmstadt.de/news/specjms2007Results_HornetQ.html

HornetQ core

HornetQ core is designed simply as a set of POJOs.

It has also been designed to have as few dependencies on external jars as possible.
As a result HornetQ core has only one more jar dependency than the standard JDK classes; netty.jar
netty buffer classes are used internally.

Each HornetQ server has its own ultra high performance **persistent journal**, which it uses for messaging and other persistence.

Using a high performance journal allows persistence message performance, which is something not achievable when using a relational database for persistence.

HornetQ modes

HornetQ currently provides two APIs for messaging at the client side:

Core client API

simple intuitive Java API that allows the full set of messaging functionality without some of the complexities of JMS.

JMS client API

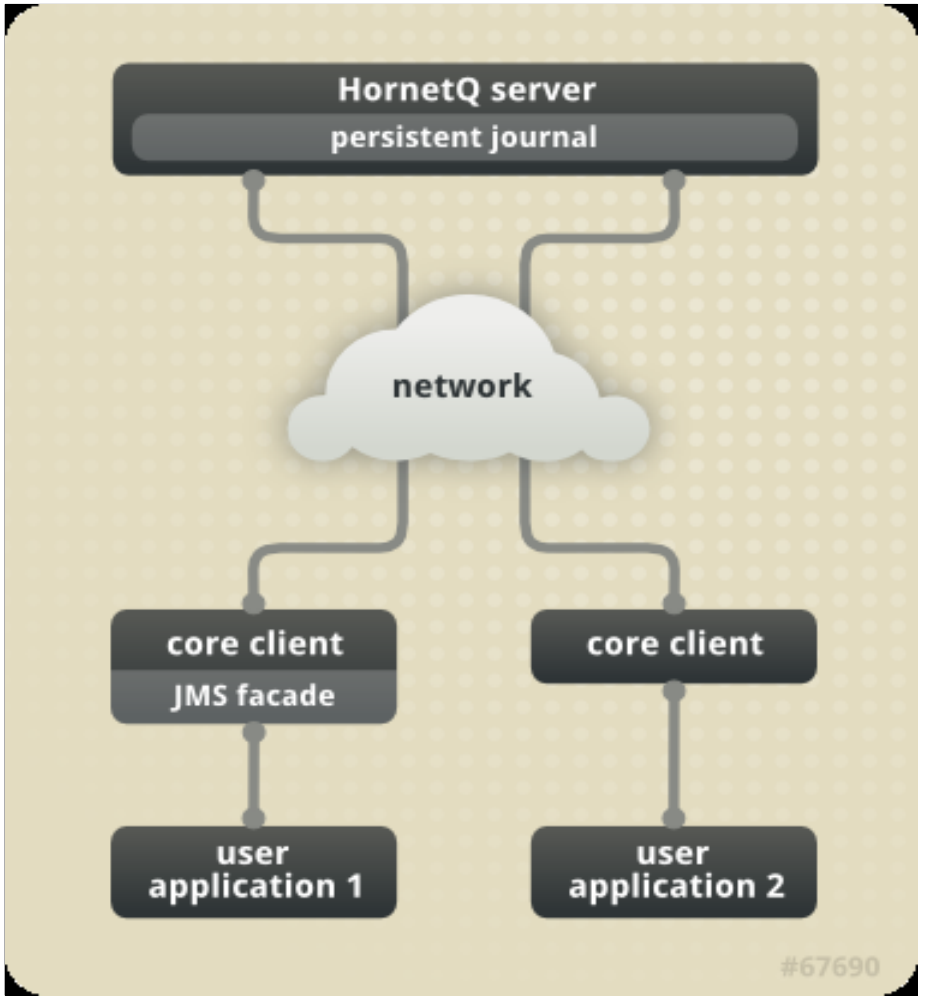
standard JMS API

JMS semantics are implemented by a thin JMS facade layer on the client side.

The HornetQ server does not associate with JMS and does not know anything about JMS. It is a **protocol agnostic messaging server** designed to be used with multiple different protocols.

When a user uses the JMS API on the client side, all JMS interactions are translated into operations on the HornetQ core client API before being transferred over the wire using the HornetQ wire format.

HornetQ modes



HornetQ in EAP

JBoss EAP 5.1.x

- not the default messaging system
- easy to switch from JBM to HornetQ
- JBM or HornetQ not together!

JBoss EAP 6.x

- default JMS implementation

HornetQ configuration in JBoss EAP 5.1.x

jboss/$profile/deploy/hornetq

- hornetq-configuration.xml
- hornetq-jms.xml

HornetQ configuration in JBoss EAP 6.x

jboss/standalone/configuration

- standalone-*.xml

jboss/domain/configuration

- domain.xml

HornetQ in EAP 6

```
<security-settings>
        <security-setting match="#">
            <permission type="send" roles="guest"/>
            <permission type="consume" roles="guest"/>
            <permission type="createNonDurableQueue" roles="guest"/>
            <permission type="deleteNonDurableQueue" roles="guest"/>
         </security-setting>
  </security-settings>
  <address-settings>
        <address-setting match="#">
            <dead-letter-address>jms.queue.DLQ</dead-letter-address>
            <expiry-address>jms.queue.ExpiryQueue</expiry-address>
            <redelivery-delay>0</redelivery-delay>
           <max-size-bytes>10485760</max-size-bytes>
            <address-full-policy>BLOCK</address-full-policy>
            <message-counter-history-day-limit>10</message-counter-history-day-limit>
         </address-setting>
</address-settings>
```

security

addressing

HornetQ in EAP 6

```
<jms-connection-factories>
        <connection-factory name="InVmConnectionFactory">
            <connectors>
                    <connector-ref connector-name="in-vm"/>
                </connectors>
                <entries>
                    <entry name="java:/ConnectionFactory"/>
                </entries>
            </connection-factory>
            <connection-factory name="RemoteConnectionFactory">
                <connectors>
                    <connector-ref connector-name="netty"/>
                </connectors>
                <entries>
                    <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/>
                </entries>
            </connection-factory>
```

connection factories

HornetQ in EAP 6

```
<pooled-connection-factory name="hornetq-ra">
        <transaction mode="xa"/>
            <connectors>
                <connector-ref connector-name="in-vm"/>
            </connectors>
            <entries>
                <entry name="java:/JmsXA"/>
            </entries>
        </pooled-connection-factory>
    </jms-connection-factories>
    <jms-destinations>
            <jms-queue name="notificationQueue">
              <entry name="/queue/notificationQueue"/>
        </jms-queue>
    </jms-destinations>
</hornetq-server>
```

jms destinations

HornetQ in EAP 6

```
public void sendToQueue(String destinationName,Serializable payload) throws Exception {
      InitialContext ic = new InitialContext();
      ConnectionFactory cf = (ConnectionFactory)ic.lookup("/ConnectionFactory");
      Queue queue = (Queue)ic.lookup(destinationName);
      Connection connection = cf.createConnection();
      Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      MessageProducer publisher = session.createProducer(queue);
      connection.start();

      ObjectMessage message = session.createObjectMessage(payload);
      message.setObject(payload);
      publisher.send(message);

      if (connection != null)   {
            connection.close();
      }
}
```

jms producer

!!!Zero-code change from JBM!!!

```
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
public void onMessage(Message message) {
      ObjectMessage obj = (ObjectMessage) message;
      try {
            Serializable ser = obj.getObject();
            log.info("[NotificationInbound] onMessage!");
      }
      catch (Exception e) {
            log.error("[NotificationInbound] ERROR[" + e.getMessage() + "]!!!****");
            throw new IllegalStateException();
      }
}
```

jms consumer

HornetQ standalone

```
hornetq
 |___ bin
      |
      config
      |      |___ jboss-as-4
      |      |___ jboss-as-5
      |      |___ stand-alone
      |
      docs
      |      |___ api
      |      |___ quickstart-guide
      |      |___ user-manual
      |
      examples
      |      |___ core
      |      |___ javaee
      |      |___ jms
      |
      lib
      |
      licenses
      |
      schemas
```

HornetQ transport

HornetQ has a fully pluggable and highly flexible transport layer. The transport layer defines its own Service Provider Interface (SPI) to simplify plugging in a new transport provider.

Netty TCP
Netty SSL
Netty HTTP
Netty Servlet

acceptors are used on the server to define how connections are accepted

hornetq-configuration.xml

```
<acceptor name="netty">
   <factory-class>
org.hornetq.core.remoting.impl.netty.NettyAcceptorFactory</factory-class>
   <param key="host"  value="${jboss.bind.address:localhost}"/>
   <param key="port"  value="${hornetq.remoting.netty.port:5445}"/>
</acceptor>
```

HornetQ transport

connectors are used by a client to define how it connects to a server

hornetq-configuration.xml

```
<connector name="netty">
  <factory-class>org.hornetq.core.remoting.impl.netty.NettyConnectorFactory</factory-class>
  <param key="host" value="${jboss.bind.address:localhost}"/>
  <param key="port" value="${hornetq.remoting.netty.port:5445}"/>
</connector>
```

HornetQ persistence

HornetQ handles persistence with a high-performance journal, which is optimized for messaging-specific use cases.

The HornetQ journal is append only with a configurable file size, which improves performance by enabling single write operations. It consists of a set of files on disk, which are initially pre-created to a fixed size and filled with padding.

As server operations (add message, delete message, update message, etc.) are performed, records of the operations are appended to the journal until the journal file is full, at which point the next journal file is used.

HornetQ persistence

configuration in **hornetq-configuration.xml**

journal-directory

location of the message journal. default value is data/journal.

journal-type

valid values are NIO or ASYNCIO.
If NIO, the Java NIO journal is used.
If ASYNCIO, Linux asynchronous IO is used. If ASYNCIO is set on a non-Linux or non-libaio system, HornetQ detectsthis and falls back to NIO.

journal-sync-transactional

If true, HornetQ ensures all transaction data is flushed to disk on transaction boundaries(commit, prepare, and rollback).
default is true.

HornetQ persistence

journal-file-size

size of each journal file in bytes.
default value is 10485760 bytes (10 megabytes).

journal-min-files

minimum number of files the journal maintains

journal-max-io

maximum number of write requests to hold in the IO queue. Write requests are queued here before being submitted to the system for execution. If the queue fills, writes are blocked until space becomes available in the queue.

journal-compact-min-files

minimum number of files before the journal will be compacted.
default value is 10.

HornetQ flow control

Flow control is used to limit the flow of data between a client and server, or a server and another server. It does this in order to prevent the client or server being overwhelmed with data.

Consumer flow control

HornetQ consumers improve performance by buffering a certain number of messages in a client-side buffer before passing them to be consumed.

By default, the consumer-window-size is set to 1 MiB

The value can be:

- -1 for an unbound buffer
- 0 to not buffer any messages.
- >0 for a buffer with the given maximum size in bytes.

HornetQ flow control

configuration in **hornetq-jms.xml**

```
<connection-factory name="ConnectionFactory">
 <connectors>
  <connector-ref connector-name="netty-connector"/>
 </connectors>
 <entries>
  <entry name="ConnectionFactory"/>
 </entries>
 <consumer-window-size>0</consumer-window-size>
</connection-factory>
```

HornetQ flow control

It is also possible to control the rate at which a consumer can consume messages.
This can be used to make sure that a consumer never consumes messages at a rate faster than the rate specified.

```
<connection-factory name="ConnectionFactory">
 <connectors>
  <connector-ref connector-name="netty-connector"/>
 </connectors>
 <entries>
  <entry name="ConnectionFactory"/>
 </entries>
 <consumer-max-rate>10</consumer-max-rate>
</connection-factory>
```

HornetQ flow control

It is possible to manage the flow control even for producers!

HornetQ also can limit the amount of data sent from a client to a server to prevent the server being overwhelmed.

```
<connection-factory name="NettyConnectionFactory">
   <connectors>
     <connector-ref connector-name="netty-connector"/>
   </connectors>
   <entries>
      <entry name="/ConnectionFactory"/>
   </entries>
   <producer-window-size>10</producer-window-size>
</connection-factory>
```

HornetQ message redelivery

An undelivered message returns to the queue ready to be redelivered.

There are two options for these undelivered messages:

Delayed Redelivery

Message delivery can be delayed to allow the client time to recover from transient failures and not overload its network or CPU resources.

Dead Letter Address

Configure a dead letter address, to which messages are sent after being determined undeliverable.

HornetQ message redelivery

```
<address-setting match="jms.queue.exampleQueue">
   <redelivery-delay>5000</redelivery-delay>
</address-setting>
```

```
<address-setting match="jms.queue.exampleQueue">
   <dead-letter-address>jms.queue.deadLetterQueue</dead-letter-address>
   <max-delivery-attempts>3</max-delivery-attempts>
</address-setting>
```

HornetQ large messages

HornetQ supports sending and receiving of large messages, even when the client and server are running with limited memory.

As the InputStream is read, the data is sent to the server as a stream of fragments. The server persists these fragments to disk as it receives them. When the time comes to deliver them to a consumer they are read back off the disk, also in fragments, and sent down the wire.

When the consumer receives a large message it initially receives just the message with an empty body. It can then set an OutputStream on the message to stream the large message body to a file on disk or elsewhere.
At no time is the entire message body stored fully in memory, either on the client or the server.

HornetQ large messages

hornetq-configuration.xml

```
<configuration xmlns="urn:hornetq"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:hornetq /schema/hornetq-configuration.xsd">
        ...
        <large-message-directory>/data/large-messages</large-message-directory>
        ...
</configuration>
```

hornetq-jms.xml

```
<connection-factory name="ConnectionFactory">
        <connectors>
                <connector-ref connector-name="netty"/>
        </connectors>
        <entries>
                <entry name="ConnectionFactory"/>
                <entry name="XAConnectionFactory"/>
        </entries>
        <min-large-message-size>250000</min-large-message-size>
</connection-factory>
```

HornetQ large messages

hornetq-configuration.xml

```
<configuration xmlns="urn:hornetq"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:hornetq /schema/hornetq-configuration.xsd">
        ...
        <large-message-directory>/data/large-messages</large-message-directory>
        ...
</configuration>
```

hornetq-jms.xml

```
<connection-factory name="ConnectionFactory">
        <connectors>
                <connector-ref connector-name="netty"/>
        </connectors>
        <entries>
                <entry name="ConnectionFactory"/>
                <entry name="XAConnectionFactory"/>
        </entries>
        <min-large-message-size>250000</min-large-message-size>
</connection-factory>
```

HornetQ paging

HornetQ transparently supports huge queues containing millions of messages while the server is running with limited memory.

In such a situation it's not possible to store all of the queues in memory at one time, so HornetQ transparently pages messages in and out of memory as they are needed. This allows massive queues with a low memory footprint.

HornetQ will start paging messages to disk when the size of all messages in memory for an address exceeds a configured maximum size.

By default, HornetQ does not page messages; this must be explicitly configured to activate it.

HornetQ paging

hornetq-configuration.xml

<paging-directory>/somewhere/paging-directory</paging-directory>

<address-settings>

<address-setting match="jms.someaddress">

<max-size-bytes>104857600</max-size-bytes>

<page-size-bytes>10485760</page-size-bytes>

<address-full-policy>PAGE</address-full-policy>

</address-setting>

</address-settings>

Address full policy:

PAGE further messages will be paged to disk.

DROP then further messages will be silently dropped.

BLOCK then client message producers will block when they try and send further messages.

HornetQ high availabilty

HornetQ allows pairs of servers to be linked together as live - backup pairs.

A backup server is owned by only one live server.
Backup servers are not operational until failover occurs.

When a live server crashes or is brought down in the correct mode, the backup server currently in passive mode will become live and another backup server will become passive. If a live server restarts after a failover then it will have priority and be the next server to become live when the current live server goes down, if the current live server is configured to allow automatic failback then it will detect the live server coming back up and automatically stop.

HornetQ high availabilty

configure the live and backup server to share their store, configure both **hornetq-configuration.xml**

<shared-store>true</shared-store>

Additionally, the backup server must be flagged explicitly as a backup:

<backup>true</backup>

In order for live - backup pairs to operate properly with a shared store, both servers must have configured the location of journal directory to point to the same shared location

HornetQ clustering

HornetQ clusters allow groups of HornetQ servers to be grouped together in order to share message processing load. Each active node in the cluster is an active HornetQ server which manages its own messages and handles its own connections.

hornetq-configuration.xml

for each node set the parameter **clustered to true**

Server discovery is a mechanism by which servers can propagate their connection details to:

Messaging clients. A messaging client wants to be able to connect to the servers of the cluster without having specific knowledge of which servers in the cluster are up at any one time.

Other servers. Servers in a cluster want to be able to create cluster connections to each other without having prior knowledge of all the other servers in the cluster.

Server discovery uses User Datagram Protocol (UDP) multicast to broadcast server connection settings.

HornetQ clustering

hornetq-configuration.xml

```
<discovery-groups>
   <discovery-group name="my-discovery-group">
      <local-bind-address>172.16.9.7</local-bind-address>
      <group-address>231.7.7.7</group-address>
      <group-port>9876</group-port>
      <refresh-timeout>10000</refresh-timeout>
   </discovery-group>
</discovery-groups>

<connection-factory name="ConnectionFactory">
   <discovery-group-ref discovery-group-name="my-discovery-group"/>
    <entries>
       <entry name="/ConnectionFactory"/>
    </entries>
</connection-factory>
```

HornetQ clustering

In case the connection is not downloaded by JNDI...

```
final String groupAddress = "231.7.7.7";

final int groupPort = 9876;

ConnectionFactory jmsConnectionFactory =
   HornetQJMSClient.createConnectionFactory(groupAddress, groupPort);

Connection jmsConnection1 = jmsConnectionFactory.createConnection();

Connection jmsConnection2 = jmsConnectionFactory.createConnection();
```

HornetQ clustering

Server Side load balancing

hornetq-configuration.xml

```
<cluster-connections>
   <cluster-connection name="my-cluster">
      <address>jms</address>
      <retry-interval>500</retry-interval>
      <use-duplicate-detection>true</use-duplicate-detection>
      <forward-when-no-consumers>false</forward-when-no-consumers>
      <max-hops>1</max-hops>
      <discovery-group-ref discovery-group-name="my-discovery-group"/>
   </cluster-connection>
</cluster-connections>
```

HornetQ clustering

Client Side load balancing

hornetq-jms.xml

```
<connection-factory name="ConnectionFactory">
   <discovery-group-ref discovery-group-name="my-discovery-group"/>
   <entries>
      <entry name="/ConnectionFactory"/>
   </entries>
   <ha>true</ha>
   <connection-load-balancing-policy-class-name>
 org.hornetq.api.core.client.loadbalance.RandomConnectionLoadBalancingPolicy
   </connection-load-balancing-policy-class-name>
</connection-factory>
```

HornetQ other features

Routing messages with wildcards
e.g. queue is created with an address of *queue.news.#*
queue.news.europe or *queue.news.usa* or *queue.news.usa.sport*

Message expire
HornetQ will not deliver a message to a consumer after it's time to live has been exceeded.
If the message hasn't been delivered before the time to live is reached, the server can discard it.

```
// message will expire in 5000ms from now
message.setExpiration(System.currentTimeMillis() + 5000);
```

Expiry-address

```
<!-- expired messages in exampleQueue will be sent to the expiry address expiryQueue -->
<address-setting match="jms.queue.exampleQueue">
<expiry-address>jms.queue.expiryQueue</expiry-address>
</address-setting>
```

HornetQ other features

Scheduled messages

TextMessage message = session.createTextMessage("MSG");

message.setLongProperty("_HQ_SCHED_DELIVERY", System.currentTimeMillis() + 5000);

producer.send(message);

...

// message will not be received immediately but 5 seconds later

TextMessage messageReceived = (TextMessage) consumer.receive();

Message group

Message groups are sets of messages that have the following characteristics:

• Messages in a message group share the same group id; that is, they have the same group identifier property (JMSXGroupID for JMS, _HQ_GROUP_ID for HornetQ Core API).

• Messages in a message group are always consumed by the same consumer, even if there are many consumers on a queue. They pin all messages with the same group id to the same consumer.

If that consumer closes another consumer is chosen and will receive all messages with the samegroup id.

HornetQ other features

Based on message

```
Message message = ...
message.setStringProperty("JMSXGroupID", "Group-0");
producer.send(message);
message = ...
message.setStringProperty("JMSXGroupID", "Group-0");
producer.send(message);
```

Based on connection factory...

```
<connection-factory name="ConnectionFactory">
<connectors>
<connector-ref connector-name="netty-connector"/>
</connectors>
<entries>
<entry name="ConnectionFactory"/>
</entries>
<group-id>Group-0</group-id>
</connection-factory>
```

hornetq-jms.xml

HornetQ other features

Diverts

Diverts are objects that transparently divert messages routed to one address to some other address,without making any changes to any client application logic.

Diverts can also be configured to apply a **Transformer**.

An **exclusive divert** diverts all matching messages that are routed to the old address to the new address.
Matching messages do not get routed to the old address.

hornetq-configuration.xml

```
<divert name="prices-divert">
<address>jms.topic.priceUpdates</address>
<forwarding-address>jms.queue.priceForwarding</forwarding-address>
<filter string="office='New York'"/>
<transformer-class-name>
org.hornetq.jms.example.AddForwardingTimeTransformer
</transformer-class-name>
<exclusive>true</exclusive>
</divert>
```

HornetQ other features

Diverts

Non-exclusive diverts forward a copy of a message to a new address, allowing the original message to continue to the previous address.

hornetq-configuration.xml

```
<divert name="order-divert">
      <address>jms.queue.orders</address>
      <forwarding-address>jms.topic.spyTopic</forwarding-address>
      <exclusive>false</exclusive>
</divert>
```

JBoss by Red Hat

Faster & Greater Messaging System
HornetQ zzz

THANKS!!!

HornetQ